Efficient Computation of Optimal Thresholds in Cloud Auto-scaling Systems

Thomas Tournaire, Hind Castel-Taleb, Emmanuel Hyon

OPTIMAL CONTROL POLICIES FOR RESOURCE ALLOCATION IN THE CLOUD: COMPARISON BETWEEN MARKOV DECISION PROCESS AND HEURISTIC APPROACHES

Thomas Tournaire [*†] Hind Castel-Taleb [†] Emmanuel Hyon [‡§]

May 3, 2021

ABSTRACT

We consider an auto-scaling technique in a cloud system where virtual machines hosted on a physical node are turned on and off depending on the queue's occupation (or thresholds), in order to minimise a global cost integrating both energy consumption and performance. We propose several efficient optimisation methods to find threshold values minimising this global cost: local search heuristics coupled with aggregation of Markov chain and with queues approximation techniques to reduce the execution time and improve the accuracy. The second approach tackles the problem with a Markov Decision Process (MDP) for which we proceed to a theoretical study and provide theoretical comparison with the first approach. We also develop structured MDP algorithms integrating hysteresis properties. We show that MDP algorithms (value iteration, policy iteration) and especially structured MDP algorithms outperform the devised heuristics, in terms of time execution and accuracy. Finally, we propose a cost model for a real scenario of a cloud system to apply our optimisation algorithms and show their relevance.

1 Introduction

Rapid growth of the demand for computational power by scientific, business and web-applications has led to the creation of large-scale data centers which are often oversized to ensure the quality of service of hosted applications. This leads to an under-utilisation of the servers which implies important electric losses and over-consumption. Currently, Cloud computing requires more electric power than those of whole countries such as India or Germany [1, 2].

To improve the utilisation rate of servers, some data center owners have deployed some methods implementing dynamicity of resources according to system load. Those mechanisms, called "*autoscaling*" [3] are based on activation and deactivation of Virtual Machines (VMs) [4] according to the workload.

However, finding the policy that tailors resources to demand is a crucial point that requires accurate assessment of both the energy expended and the performance of the system. Unfortunately, these two measures are inversely proportional, which motivates researchers to evaluate them simultaneously via a unique global cost function. One way to adapt the capacities is hysteresis models which make it possible to correctly represent the variability of resources according to the demand [5] since the hysteresis phenomenon allows the cloud owner not to activate and deactivate too frequently when the load is varying. Hence, multi-server threshold based-queueing systems with hysteresis policy have been proposed to efficiently manage the number of active VMs [6, 7, 8]. The other advantage of hysteresis policies is that they are a key component of the auto-scaling systems for Azure and Amazon Web Services [3]. Henceforth, there is a great interest of studying the computation of hysteresis policy since the threshold values are these one that can be plugged into the autoscaling system that are implemented in the major cloud architecture.

[*]Nokia Bell Labs France, thomas.tournaire@nokia.com.
[†]Samovar, Telecom SudParis.
[‡]Sorbonne Université, UPMC Univ Paris 06, CNRS, LIP6 UMR 7606.
[§]Université Paris Nanterre.

In this work, we consider a cost-aware approach and we propose a model with global expected cost considering different costs: associated with performance defined in a Service Level Agreement (SLA) and energy consumption.

We propose to minimise this global cost using two modelling approaches: static optimisation problem or dynamic control. The static optimisation problem expresses the problem by using the average value of the costs computed by stationary distribution of a CTMC (Continuous Time Markov Chains) while dynamic control expresses the problem under a MDP (Markov Decision Process) model. The efficiency of these two approaches are not assessed very well in the literature for the computation of single thresholds by level and had been never done for hysteresis policies. Determining which is the best promising approach is thus a major point.

The key contributions of this paper are as follows:

1. We improve the existing heuristics of [9] that makes a large review of the heuristics of the literature which solve static optimisation problems. We provide an algorithm that solves dynamic control model by considering hysteresis assumption in the MDP. We assess numerically the gain given by these improvements.
2. We provide a theoretical study of the multichain properties of the Markov decision process. This shows that some usual algorithms solving MDP model have no convergence guarantee;
3. We made a theoretical analysis between the two approaches and give some insights which explain why the static optimisation problem is suboptimal. Numerical studies show that the dynamic control approach strongly outperforms the other one for optimality and running time criteria;
4. We develop and analyses a financial cost model which takes into account energy consumption of VMs and prices of instances in cloud providers. Presentation of minimised cost values problem based on this concrete cloud model.

The remainder of this paper is organised as follows. Section 2 briefly reviews the related works. In Section 3, we provide a treatment to unify the different definitions of hysteresis policies coming from different models. Then, we describe the cloud system, with the queueing model and the cost function used to express the expected costs in terms of performance and energy consumption for the model. In Section 4, we present the Markov chain approach with the decomposition and aggregation method, and the optimisation algorithms based on local search heuristics. We also present a Markov decision process model and adapted algorithms in Section 5. Section 6 presents a concrete cost model, and computational experiences with comparison between algorithms are discussed in Section 7. Finally, achieved results are discussed in the conclusion and comments about further research issues are given.

2 Literature review

2.1 Energy and Performance Management in the Cloud

The cost of energy is one of the many challenges facing large-scale computing. When the resource utilisation is too low, some of them can be turned off to save energy, without sacrificing performance.

In a data center, the power consumption can be divided into static and dynamic parts. The static parts are the base costs of running the data center when being idle and the dynamic costs depend on the current usage. In [10], they define a power-aware model to estimate the dynamic part of energy cost for a VM of a given size, this model keeps the philosophy of the pay as you go model but based on energy consumption.

Two main approaches of physical server resource management have been proposed to improve the energy efficiency: shutdown or switching on servers or VMs which is referred as dynamic power management [4], and scaling of the CPU performance referred as Dynamic Voltage and Frequency Scaling [11]. Shutdown strategies (considered here) are often combined with consolidation algorithms that gather the load on a few servers to favour the shutdown of the others. Hence, managing energy by switching on or switching off VM is an intuitive and fairly widespread way to save energy. However, coarse techniques of shutdown are not necessarily the best solution to achieve energy reduction as quoted in [4]. Indeed the main disadvantage of shutdown policies resides in the energy and time losses that may occur when switching off and on takes longer than the idle period.

2.2 Control Management for Queueing Models

Server farm models [12, 13, 14] have been proposed to represent these problems with activations and deactivations of virtual machines. These server farm models are a good representation of the dynamicity in a data center. They are often modelled with multi-server queueing models [15, 16] which allows to easily compute performance metrics. Multi-server models do not address issues related to the internal network of the cloud or the VM placement in it. Instead,

all VM are considered as parallel resources which make these multi-server models appropriate for studying simple nodes of several servers in the cloud.

However, the way with which one decides to switch on or switch off the VM remains a key point and the computation of the optimal action has led to a large field of researches. One can search the optimal policy using different methods such as dynamic control (and Markov decision Process) or otherwise search among policies having a special form what makes them more tractable in practice.

2.3 Markov Decision Process and Hysteresis Policies

Since the seminal work of Mc Gill in 1969 (with an optimal control model) and that of Lippman (with a Markov decision process model) numerous works have been devoted to similar multi-server queue models using Markov decision processes (see [17] and references therein for the oldest, and more recently [18] and [19] to quote just a few). Unfortunately, not all of them received rigorous treatment and the study of unichain or multichain property is often ignored. It appeared very early, in the work of Bell for a $M/M/2$ model (see [17]), that the optimal policy in such models has a special form and is called *hysteresis*. In hysteresis policies, servers are powered up when the number of jobs in the system is sufficiently high, and are powered down when that number is sufficiently low. More precisely activations and deactivations of servers are governed by sequences of different forward and reverse thresholds. The concept of hysteresis can also be applied to the control of service rates.

The axis of researches for hysteresis policies are twofold: the first one is the exploration of the conditions to insure the optimality of hysteresis policy. Szarkowicz et al. [20] showed the optimality of hysteresis policies in $M/M/S$ queueing model with a control of the vacations of the servers. For models with control of the service rates, the proof are made in Hipp or Serfozo [6]. The second axis studies the computation of the threshold values.

2.4 Threshold calculation methods

The search for optimal thresholds received less attention in the past, but also here two major trends appeared: either computation of the optimal policy by mean of adapted usual dynamic programming algorithms (also by considering structured policies) or optimisation methods based on the computation of expected measures associated with a set of thresholds. In [21], Song claims that, for single threshold models, the second approach dealing with stationary distribution computations are generally more effective than the MDP approach. Such a comparison has not been performed yet for hysteresis especially since no work (up to our knowledge) implements a MDP algorithm with a structured policy.

The stationary distribution computation is difficult since cloud systems are often defined on very large state spaces. Nevertheless, under some assumptions, evaluation of hysteresis multi-server systems has been already studied in the literature and different resolution methods have been presented to compute efficiently the performance measures of the system. Among the most significant works, we can mention the work of Ibe and Keilson [22] refined in Lui and Golubchik [7] which solves the model by partitioning the state space in disjoint sets in order to aggregate the Markov chain. Exhaustive comparisons of the resolution methods are made in [23]. Closed-form solution of the stationary distribution, partition of the state space and matrix geometric methods applied on QBD (Quasi birth and death) processes are studied. It can be seen that partitioning is the most suited. Furthermore, for optimisation, the objective cost function being non linear and non convex makes this problem very complex and there is currently no exact method to solve this problem. Only some approximate heuristics are known but they require the computation of the steady state and the costs for several threshold values which requires very high computation times. The work [24] presents three heuristics for searching single thresholds in inventory models that have been adapted for hysteresis in [9]. For a server farm model with activation of a reserve, Mitrani [14] uses fluid approximation to compute the activation thresholds and [25] uses genetic algorithm for optimisation and matrix geometric method for the stationary distribution computation.

3 Cloud Model

In this paper, we focus on IaaS clouds where virtualisation has enabled the abstraction of computing resources so that a physical server is able to work as a set of multiple logical Virtual Machines (VMs). Next, we present the multi-servers queueing model that manages these multiple logical VMs and which is the queueing model used to analyse mathematically performance and energy consumption.

3.1 Controlled multi-server queue

We consider a controlled queueing model, denoted *Cloud Model*, representing a physical server of a data center. The system is a multi-server queueing system where each server represents a Virtual Machine (VM) (or evenly a core). We have the following assumptions for the model:

1. Requests arrive in the system following a Poisson process of rate λ, and service times of all VMs are independent of arrivals and independent of each other. Moreover they are i.i.d. and follow an exponential distribution with an identical rate μ;
2. The queue has a finite size B, requests are served by a set of VMs, whose total number is K, and the service discipline is FIFO (First In First Out).

A customer is treated by a server as soon as this server becomes idle and the server is active. Servers can be turned on and off by the controller. When the server is turned on it becomes active while it becomes inactive when it is turned off.

We define $\mathcal{S}$ the state space, where $\mathcal{S} = \{0, 1, \ldots, B\} \times \{1, \ldots, K\}$. Any state $x \in \mathcal{S}$ is such that $x = (m, k)$ where m represents the number of customers in the system, and k is the number of operational servers (or active servers, this number can also be seen as a service level). We define $\mathcal{A} = \{0, \ldots, K\}$, be the set of actions, where action a denotes the number of servers to be activated. With this system, are associated two kinds of costs that a cloud provider encounters:

1. Costs corresponding to the performance of the system, for the control of the service quality defined in the SLA: as costs (C_H) per unit of time for holding requests in the queue or instantaneous costs (C_R) for losses of requests.
2. Costs corresponding to the use of resources (operational and energy consumption): as costs for using a VM per unit of time (C_S) and instantaneous costs for activating/deactivating (C_A and C_D).

We define for the system a global cost taking into account both performance and energy consumption costs. We have the following objective function we want to minimise:

$$\bar{C} = \lim_{T \to \infty} \frac{1}{T} \mathbb{E}\left\{ \int_0^T C_t \left(X(t), A(t) \right) dt \right\} \tag{1}$$

where $A(t)$ is the action taken at time t (it is possible that nothing was performed) and $C_t(X(t), A(t))$ is the cost that is charged over time when the state is $X(t)$ and action $A(t)$ is performed. This problem can be solved using dynamic programming algorithms or by computing the corresponding thresholds values for turning on and off the VMs.

3.2 Models of hysteresis policies

A decision rule is a mapping from some information set to some action and we call policy a sequence of decision rules $\eta = (q_0, q_1, q_2, \ldots)$. While the most general set of policies is that of history-dependent randomised policies, the classical results on average infinite-horizon, time-homogeneous Markovian optimal control [26] allow us to concentrate on stationary Markov Deterministic Policies. Such policies are characterised by a single, deterministic decision rule which maps the current state to a decision. We thus consider the mapping q from $\mathcal{S}$ to $\mathcal{A}$ such that $q(x) = a$.

All along the paper we restrict our attention with a special form of policies: the *hysteresis* policies. Nevertheless, there is relatively few homogeneity between definitions of hysteresis policies in the literature. Indeed, it can refer to policies defined with double thresholds (especially when $K = 2$), or to a restrictive definition when Markov chains are used. We follow the works of [27, 6] to present an unified treatment of hysteresis.

Hysteresis policies with multiple activations We assume in this part that the decision rule is a mapping from the state to a number of active servers $q(m, k) = k_1$ with $k_1 \in [1, \ldots, K]$. Several servers can be activated or deactivated to pass from k to k_1 active servers.

Definition 1 (Double threshold policies)**.** *We call* double threshold *policy a stationary policy such that the decision rule $q(m, k)$ is increasing in both of its arguments and is defined by a sequence of thresholds such that for any k and k_1 in $[1, K]$ we have:*

$$q(m, k) = k_1 \text{ when } \quad \ell_{k_1}(k) \leq m < \ell_{k_1+1}(k)\,.$$

where $\ell_{k_1}(k) = \min\{m : q(m, k) \geq k_1\}$. This minimum is ∞ if the set is empty. For all k, we also fix $\ell_{K+1}(k) = \infty$ and $\ell_1(k) = 0$ (since at least one server must be active).

A monotone hysteresis policy is a special case of double threshold policy.

Definition 2 (Monotone Hysteresis polices [27])**.** *A policy is a monotone hysteresis policy if it is a double threshold policy and moreover if it exists two sequences of integers l_k and L_k such that*

$$l_k = \ell_k(K) = \ell_k(K-1) = \ldots = \ell_k(k)$$
$$L_k = \ell_k(1) = \ell_k(2) = \ldots = \ell_k(k-1)\,,$$

with $l_k \leq L_k$ for $k = 1, \ldots, K+1$; $l_k \leq L_{k+1}$ for $k = 1, \ldots, K$; $l_1 = L_1 = 0$, $l_{K+1} = L_{K+1} = \infty$.

$$\text{And if, for all } (m,k) \in \mathcal{S},\ q(m,k) = \begin{cases} q(m,k-1) & \text{if } m < l_k \text{ and } k > 1 \\ k & \text{if } l_k \leq m < L_{k+1} \\ q(m,k+1) & \text{if } m \geq L_{k+1} \text{ and } k < m \end{cases}$$

The thresholds l_k can be seen as the queue levels at which some servers should be deactivated and the L_{k+1} are the analogous activation points. Roughly speaking, the difference between a double threshold and a monotone hysteresis policy lies in the fact that some thresholds toward a level are identical in hysteresis.

Definition 3 (Isotone Hysteresis polices [27])**.** *] A policy is an isotone policy if it is a monotone hysteresis policy and if $0 = l_1 \leq l_2 \leq \ldots \leq l_{K+1} = \infty$ and $0 = L_1 \leq L_2 \leq \ldots \leq L_{K+1} = \infty$.*

Example 1. *In Figure 1, we represent an isotone policy. The number indicates the number of servers that should be activated in each state. The bold line means that the number of activated servers is the same than the decision and then no activation or deactivation have to be performed.*

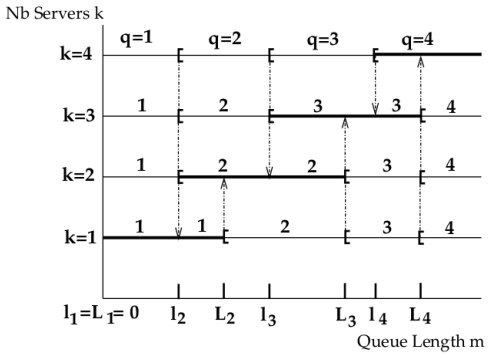

Figure 1: An isotone hysteresis policy.

Proposition 1 (Optimality of monotone hysteresis policies [20])**.** *In the multi-server queueing model presented in Section 3 and for which there is activation/deactivation costs, working costs and holding costs, Szarkowicz and al. showed that monotone hysteresis policies are optimal policies.*

However, the presence of a rejection cost, as we assume here, is not considered in the assumptions of [20] and there is no proof about the optimality of hysteresis policies for the model studied here.

Remark. There is an alternate way to express the policy by giving the number of servers to activate or deactivate instead of giving directly the number of server that should be active.

Hysteresis policies with single VM activation We focus now on models in which we can only operate one machine at a time. The decision rule indicates now an activation or deactivation. Therefore, the action space is now $\mathcal{A} = \{-1, 0, 1\}$. For a state (m, k), when the action $q \in \mathcal{A}$ is -1 then we deactivate a server, when the action is 0 then the number of active servers remains identical, when the action is 1 then we activate a server.

It could be noticed that, for this kind of model, double threshold policies and hysteresis policies coincide. Indeed, the decision now relates to activation (resp. deactivation) and no longer to the number of servers to activate (resp. deactivate). There exist only two thresholds by level k : L_{k+1} to go to level $k+1$, and l_k to go to level $k-1$, and there are no other levels than can be reached from k. For example, in Figure 1 all decisions smaller than the level are replaced by -1 while all decisions larger than the level are replaced by 1. Definition 3 remains unchanged but Definition 2 should be rephrased in:

Definition 4 (Monotone hysteresis policy)**.** *A policy is a monotone hysteresis policy if it is a stationary policy such that the decision rule $q(m,k)$ is increasing in m and decreasing in k and if is defined by two sequences of thresholds l_k and L_k such that for all $(m,k) \in \mathcal{S}$:*

$$q(m,k) = \begin{cases} -1 & \text{if } m < l_k \text{ and } k > 1 \\ 0 & \text{if } l_k \leq m < L_{k+1} \\ 1 & \text{if } m \geq L_{k+1} \text{ and } k < m \end{cases},$$

with $l_k \leq L_k$ for $k = 1, \ldots, K+1$; $l_k \leq L_{k+1}$ for $k = 1, \ldots, K$ and $l_1 = L_1 = 0$; $l_{K+1} = L_{K+1} = \infty$.

Hysteresis policies and Markov chain As presented in [17], there is a slightly different model of multi-server queue with hysteresis policy which received a lot of attention (see Ibe and Keilson [22] or Lui and Golubchik [7] and references therein). This model is still a multi-server queueing system but is no longer a controlled model: the transitions between levels are also governed by sequences of prefixed thresholds this is why it is called an hysteresis model. It is built to be easily represented by a Markov chain. The differences between the controlled model and this Markov chain model are detailed in Section 5.3.

Remark. For convenience, deactivation and activation thresholds will be denoted respectively by R and F in the Markov chain model, while they will be denoted l and L in the MDP model.

Definition 5 (Hysteresis policy [7])**.** *A K-server threshold-based queueing system with hysteresis is defined by a sequence $F = [F_1, F_2, \ldots, F_{K-1}]$ of activation thresholds and a sequence $[R_1, R_2, \ldots, R_{K-1}]$ of deactivation thresholds. For $1 \leq k < K$, the threshold F_k makes the system goes from level k to level $k+1$ when a customer arrives with k active servers and F_k customers in the system. Conversely, the threshold R_k makes the system goes from level $k+1$ to level k when a customer leaves with $k+1$ active servers and $R_k + 1$ customers in the system.*

Furthermore, we assume that $F_1 < F_2 < \ldots < F_{K-1} \leq K$, $1 \leq R_1 < R_2 < \ldots < R_{K-1}$, and $R_k < F_k$, $\forall\, 1 \leq k \leq K-1$. We denote the vector that merges the activation threshold vector F and the deactivation threshold vector R is denoted by $[F, R]$.

Remark. It can be noticed that, no server can remain idle all the time here since the threshold values are bounded, whereas in Definition 4 when a threshold is infinite the server remains inactive. Hence the hysteresis policy presented in [22] can be seen as a restricted version of isotone policies given in 4. Furthermore, the inequalities being strict in [7], the hysteresis of Definition 5 is a very specific case of hysteresis of Definition 4 called *strictly isotone*.

4 Hysteresis policies and Markov chain approach

This section is devoted to the study of the policies defined in Definition 5 and their aggregation properties.

4.1 Hysteresis queueing model

We assume that the model of hysteresis defined in Definition 5 is used. Once the thresholds $[F, R]$ are fixed, the underlying model is described by a Continuous-Time Markov Chain (CTMC), denoted $\{X(t)\}_{t \geq 0}$. A state is represented by a couple $(m,\, k)$ such that m is the number of requests in the system and k is the operating level, corresponding to the number of active VMs. The state space is denoted by $\mathcal{X} \subset \mathcal{S}$ and is given by:

$$\mathcal{X} = \{(m,\, k) \mid 0 \leq m \leq F_1, \text{ if } k = 1\,, R_{k-1} + 1 \leq m \leq F_k, \text{if } 1 < k < K\,, R_{K-1} + 1 \leq m \leq B, \text{if } k = K\} \quad (2)$$

The transitions between states then follows:

$$\begin{aligned} (m,k) \rightarrow\ & (\min\{B, m+1\}, k), \text{ with rate } \lambda\,, \ \text{ if } m < F_k\,; \\ \rightarrow\ & (\min\{B, m+1\}, \min\{K,\, k+1\}), \text{with rate } \lambda\,, \ \text{ if } m = F_k\,; \\ \rightarrow\ & (\max\{0, m-1\}, k), \text{ with rate } \mu \cdot \min\{m, k\}, \text{ if } m > R_{k-1} + 1; \\ \rightarrow\ & (\max\{0, m-1\}, \max\{1, k-1\}) \text{ with rate } \mu \min\{k, m\}\,, \text{ if } m = R_{k-1} + 1\,. \end{aligned}$$

An example of these transitions is given in Fig. 2.a, for a number of levels $K = 3$, and B maximum requests in the system.

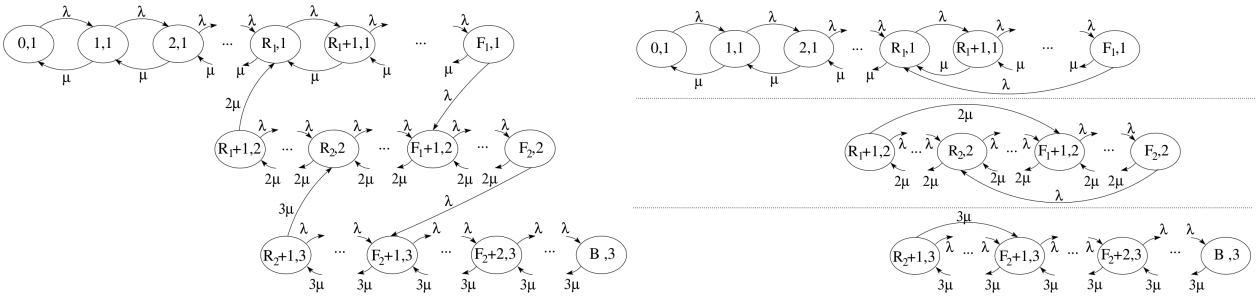

Figure 2: (a) Markov chain for K=3 , (b) Micro-chains aggregation

4.1.1 Costs

The global cost defined in Equation (1) can be rewritten as follows for fixed thresholds $[F, R]$:

$$\overline{C}^{\pi}_{[F,R]} \;=\; \sum_{k=1}^{K} \sum_{m=R_{k-1}+1}^{F_k} \pi(m,k) \cdot \overline{C}(m,k)\,, \tag{3}$$

where $\pi(m,k)$ is the stationary probability in state (m,k). For a given state of the environment (number of requests, number of activated VMs), the cloud provider pays a cost:

$$\overline{C}(m,k) = C_H \cdot m + C_S \cdot k + C_A \cdot \lambda \cdot \mathbb{1}_{m=F_k,\, k<K} + C_D \cdot \mu \cdot \min\{m,k\} \cdot \mathbb{1}_{m=R_{k-1}+1,\, 2\leq k\leq K} + C_R \cdot \lambda \cdot \mathbb{1}_{m=B,\, k=K}\,. \tag{4}$$

4.2 Decomposition and aggregation methods for calculation of the overall cost

When the number of levels K becomes very large, solving the Markov chain is complex. This is why to compute the Markov chain stationary probability distribution, we will apply a decomposition and aggregation method, using the SCA (Stochastic Complement Analysis) method [7]. The idea is to separate an initial Markov chain with K levels in K independent sub-chains, called *micro-chains* which will be solved separately. We also define an aggregated Markov chain with K states, called *macro-chain*, where each state represents an operating level.

Description of *micro-chains* and *macro-chain* Each micro-chain of a level k, such that $1 \leq k \leq K$, is built by keeping the transitions, of the initial Markov chain, inside the same level k. On the other hand, the transitions between states of different levels are cut and replaced by new transitions inside the *micro-chains* from a leaving state to an entering state of the initial chain. For example, the transitions inside the states of level 2 with rates λ and 2μ in the Fig. 2.a are kept, while the links between level 2 and level 3, and level 2 and level 1 are deleted, as it can be seen in Fig. 2.b. Then, the cut transitions between levels 2 and 3 in the initial Markov chain of Fig. 2.a are replaced by a transition between $(F_2, 2)$ and $(R_2, 2)$, with rate λ, and a transition between $(R_2 + 1, 3)$ and $(F_2 + 1, 3)$ with rate 3μ (see Fig. 2.b). The approach is similar for the other transitions between other levels and for any number of levels.

Concerning the *macro-chain*, it is described by a chain with K states, where each state k (with $1 \leq k \leq K$) represents the level. The *macro-chain* is actually a birth-death process for which the resolution is usual. We denote by Π the stationary probability distribution of the *macro-chain*. We denote by π_k the stationary probability distribution of the *micro-chain* of level k (with $1 \leq k \leq K$) and the stationary distribution of the initial Markov chain is denoted by π. We obtain the distribution of the initial chain with the following equation:

$$\forall\, (m,k) \in \mathbb{S},\; \pi(m,k) \;=\; \Pi(k) \cdot \pi_k(m)\,. \tag{5}$$

The solving of Markov chains (*micro* and *macro*) is done with a power method implemented in the *marmoteCore* framework [28] using a precision of 10^{-8}. Indeed, the use of closed formulas suffers from numerical instability [23]. Each *micro-chain* is solved independently, then the *macro-chain*, and finally the initial Markov chain. We compare the distributions obtained by the aggregation method with the ones obtained with a power method. This difference is always smaller than 10^{-8} which is the precision of our computation.

Computation of the mean cost on the aggregated chain We use the aggregation method described in Section 4.2, to aggregate the global cost $\overline{C}^{\pi}_{[F,R]}$, from Eq. (3), as a function of costs per level computed on the *micro-chains*.

Theorem 1 (Aggregated Cost Function)**.** *The mean cost $\overline{C}^{\pi}_{[F,R]}$ of the system with fixed thresholds $[F, R]$ can be written under the form:*

$$\overline{C}^{\pi}_{[F,R]} \;=\; \sum_{k=1}^{K} \Pi(k) \cdot \overline{C}(k)\,, \quad \textit{with}\;\; \overline{C}(k) = \sum_{m=R_{k-1}+1}^{F_k} \pi_k(m) \cdot \overline{C}(m,k)\,,$$

where $\overline{C}(k)$ represents the mean cost of level k.

Proof. In Eq. (3), we use Eq. (5), so we have

$$\overline{C}^{\pi}_{[F,R]} = \sum_{k=1}^{K} \sum_{m=R_{k-1}+1}^{F_k} \pi_k(m) \cdot \Pi(k) \cdot \overline{C}(m,k)\,.$$

Removing $\Pi(k)$ from the second sum (since it does not depend on m), then we obtain the overall mean cost computed from the *macro-chain*, and aggregated mean costs per level:

$$\overline{C}^{\pi}_{[F,R]} \;=\; \sum_{k=1}^{K} \Pi(k) \;\cdot \sum_{m=R_{k-1}+1}^{F_k} \pi_k(m) \cdot \overline{C}(m,k) \;=\; \sum_{k=1}^{K} \Pi(k) \cdot \overline{C}(k)$$

□

The relevance of Theorem 1 is to propose an efficient approach for the mean cost computation. Instead of computing the cost on a multidimensional Markov chain, we compute it from several one dimensional Markov chains. We take advantage of this aggregated expression coupled with a property on threshold changes, to improve the cost computation of a neighbour of a thresholds set.

Corollary 2. *Let [F,R] be a fixed vector of thresholds. It is assumed that* micro-chains *as well as costs per level are already calculated. The modification of a threshold F_k or R_k in the Markov chain has only an impact on the levels k and k+1. Therefore, calculating the average cost of the new system requires only a new computation of $\pi_k, \pi_{k+1}, \overline{C}(k), \overline{C}(k+1)$ and Π.*

The intuition about this corollary can be easily seen by looking at the Markov chain in Fig. 2.a. If we modify F_2, this will only have an impact on the distributions π_2 and π_3, thus we have to compute again $\overline{C}(2)$ and $\overline{C}(3)$, but the other mean cost $\overline{C}(1)$ is not impacted.

Proof. Recall that we define the load of the system by $\rho = \lambda/\mu$. Using the balance equations in [7], we exhibit the impact generated by the modification of activation and deactivation thresholds on the stationary probability distributions of the *micro-chains* first and then of the *macro-chain*.

Impact on the *micro-chains*: Owing to $m \in [R_{k-1}+1, F_k]$ for all $k \in \{1, \ldots, K\}$, we can therefore express the stationary probability of each state in the level k by: $\pi_k(m) \;=\; \pi_k(R_{k-1}+1) \cdot \gamma^k_m$ where:

$$\pi_k(R_{k-1}+1) = \left(\sum_{m=R_{k-1}+1}^{F_k} \gamma^k_m \right)^{-1},$$

and

$$\gamma^k_m = \begin{cases} \sum\limits_{j=0}^{m-R_{k-1}-1} \left(\frac{\rho}{k}\right)^j\,, & \text{if } R_{k-1}+1 \le m \le R_k\,; \\ \sum\limits_{j=0}^{m-R_{k-1}-1} \left(\frac{\rho}{k}\right)^j \;-\gamma^k_{F_k} \cdot \sum\limits_{j=1}^{m-R_k} \left(\frac{\rho}{k}\right)^j\,, & \text{if } R_k+1 \le m \le F_{k-1}+1\,; \\ \sum\limits_{j=m-F_{k-1}-1}^{m-R_{k-1}-1} \left(\frac{\rho}{k}\right)^j \;-\gamma^k_{F_k} \cdot \sum\limits_{j=1}^{m-R_k} \left(\frac{\rho}{k}\right)^j\,, & \text{if } F_{k-1}+2 \le m \le F_k-1\,; \\ \rho\left[\left(\frac{\rho}{k}\right)^{F_k-F_{k-1}} - \left(\frac{\rho}{k}\right)^{F_k-R_{k-1}+1}\right] \Big/ \left(k+\rho\left(\frac{\rho}{k}\right)^{F_k-R_k}\right)\,, & \text{if } m = F_k\,. \end{cases}$$

Note that $\pi_k(R_{k-1}+1)$ is determined through normalisation condition which states that the sum of probabilities of all states of the *micro-chain* of level k equals 1. We notice, in the equations above, that the stationary probability formulas of $\pi_k(m)$ of levels k only depend on the thresholds R_{k-1}, R_k, F_{k-1} and F_k. This shows that the variation of a threshold R_k or F_k has an impact only on levels k and $k+1$ but not on the other levels. Moreover, since the *micro-chain* of level k starts from state $(R_{k-1}+1, k)$ and ends in state (F_k, k), then thresholds from level 1 to $k-2$ as well as thresholds from level $k+2$ to K are not involved on the stationary distribution of the level k.

To resume, if we modify an activation threshold F_k, then it will modify states (F_k, k) and $(F_k+1, k+1)$ and the two *micro-chains* of level k and $k+1$. If we modify, a deactivation threshold R_{k-1}, then it will modify states $(R_{k-1}+1, k)$ and $(R_{k-1}, k-1)$ and the two *micro-chains* of levels $k-1$ and k.

Finally, the variation of a threshold from level k modifies only the two *micro-chains* of level k and $k+1$. Hence, to compute the cost with respect to Theorem 1 we only need to recalculate the stationary probabilities and thus the associated costs of these levels. The costs of the other levels are left unchanged.

Impact on the *macro-chain*: A change in a threshold value modifies some transitions of the *macro-chain*. This can be seen, similarly as previously, using the balance equations: $\lambda_k = \lambda \cdot \pi_k(F_k), \forall\, k = 1, \ldots, K-1$, and $\mu_k = \mu \cdot \pi_k(R_{k-1}+1), \forall\, k = 2, \ldots, K$. This gives the following formulas for the stationary probability distribution, for all $k \in \{2, \ldots, K\}$:

$$\Pi(k) = \Pi(1) \cdot \prod_{j=1}^{k-1} \left(\frac{\lambda_j}{\mu_{j+1}}\right), \text{ where } \Pi(1) = \left[1 + \sum_{k=2}^{K} \prod_{j=1}^{k-1} \left(\frac{\lambda_j}{\mu_{j+1}}\right)\right]^{-1}.$$

Since $\Pi(1)$ depends on all threshold values and since we notice that $\Pi(k)$ depends on $\Pi(1)$ for all $k \in \{2, \cdots, K\}$, therefore, when we modify a threshold F_k or R_k of any level k we need to recalculate the whole stationary probabilities of all states of the *macro-chain*. □

4.3 Thresholds calculation using stationary distributions of Markov chain

Our first approach relies on thresholds calculation using the stationary distribution of the underlined Markov chain coupled with an optimisation problem. We present now different algorithms used to compute the optimal threshold policy which minimises the mean global cost.

The optimisation problem For a set of fixed parameters: λ, μ, B, K, and costs: C_a, C_d, C_h, C_s, C_r, the algorithms aims at finding the vector of thresholds $[F, R]$, such that the objective function is minimised. The objective function is the financial cost function $\overline{C}^{\pi}_{[F,R]}$, given by Eq.(3). This mean global cost is computed knowing the stationary distribution π of the Markov chain $\{X(t)\}_{t\geq 0}$, with respect to the vectors of thresholds. Algorithms also ensure some constraints for the thresholds, based on the hysteresis principle that is: $R_k < F_k$, $R_{k-1} < R_k$ and $F_{k-1} < F_k$. Secondly, we assume that our VMs does not work for free, i.e. $k < F_k$. Thus, we want to solve the following constraint optimisation problem:

$$\begin{array}{ll} \underset{F,R}{\text{Minimise}} & \overline{C}^{\pi}_{[F,R]} \\ \text{u.c.} & R_i < F_i \quad i = 1, \ldots, K-1. \\ & F_1 < F_2 < \cdots < F_{K-1} < B \cdot \\ & 0 \leq R_1 < R_2 < \cdots < R_{K-1} \\ & F_i\ ,\ R_i \in \mathbb{N} \end{array} \tag{6}$$

This optimisation problem seems NP-hard, as the cost function is non-convex [9] and threshold values should be integer. Resolution time increases exponentially with the number of thresholds.

4.3.1 Local search heuristics

To solve this optimisation problem, several local search heuristics and one meta-heuristic have been developed in [9] for which we compared both their accuracy and execution time in numerical experiments. We kept here two heuristics **BPL**, **NLS** that have the best results and present some improvements. Most of the heuristics in [9] are inspired by the work of [24] which proposes three different heuristics to solve a lost sales inventory problem. However, unlike us, it considers only a simple threshold by class, therefore these heuristics had to be adapted to our approach, since we now have to manage double thresholds in order to return an optimal hysteresis policy of a specific cloud system.

***Best Per Level* (BPL)** This algorithm first initialises $[F, R]$ with the lowest feasible value, i.e. $F_1 = 1, F_2 = 2, ..., F_{K-1} = K$ and $R_1 = 0, R_2 = 1, ..., R_{K-1} = K - 1$. Then it improves each threshold in the following order: it starts with the first activation threshold F_1 by testing all its possible values taking into account the hysteresis constraints. A new value of F_1 that improves the global cost, will replace the old one. Then it will move on to F_2, F_3, ... , $F_{K-1}, R_1, R_2, ..., R_{K-1}$. Once a loop is finished, it will restart again until the mean global cost is not improved anymore.

***Neighbourhood Local Search* (NLS)** This algorithm is the classical local search algorithm. We randomly initialise the solution $[F, R]$. Then we generate the neighbourhood $\mathcal{V}(x)$ of a current solution x. Each neighbouring solution will be the same as the current solution, with a shift ± 1 on one of the thresholds. The algorithm explores all the neighbourhood and returns the best solution among $\mathcal{V}$. Again, it loops the same process until the mean global cost is not improved anymore.

4.3.2 Acceleration of local search algorithms using aggregation

Local search algorithms are based on the exploration inside a set of closed solutions: which have a single change of one threshold from the current solution. From Section 4.2, the computation of the global mean cost $C^{\pi}_{([F,R])}$ and the evaluation of a solution depend on the stationary distributions π_k of *micro-chains*, and Π of *macro-chain* as well as the costs per level. Thus, after the modification of a threshold we should compute all those elements again. This requires a huge amount of time which depends on the number of neighbouring solutions tested at each iteration and which increases when we consider large scale scenarios (with large K and B).

However we can ensure the algorithms to avoid so many calculations for each neighbouring solutions. Indeed the use of Corollary 2 will widely improve the algorithm's speed without impacting the efficiency by allowing us to compute only the stationary distributions of two *micro-chains* and their associated costs. The macro chain still need to be solved, but we no longer need to compute all the *micro-chain* distributions. Hence, at each iteration, we only need to compute the stationary distributions and the average costs of the two impacted *micro-chains* k and $k + 1$ among the K *micro-chains*.

Nevertheless, some algorithmic tricks should be used in order to ensure a proper running of the method. Indeed, during the study of a neighbour we have to store the two modified micro-chains of the current solutions since they will be used many times. The modification of local search heuristics with the help of aggregation reduces the number of computations to perform and then the running time. Moreover, we can conceive that this method will be more effective as K increases.

4.3.3 Solution using queueing model approximations.

We work here on a new heuristic which calculates a near-optimal solution very quickly. It can be considered as a method in its own right but can be also used for the initialisation step in local search heuristics. This heuristic is called ***MMK approximation*** since it uses $M/M/k/B$ queues to compute costs of these fairly close models.

Principle of the heuristic The main idea of the heuristic is that, at each level k we compute an approximation of the mean global cost by using the stationary distribution of the Markov chain of a $M/M/k/B$ queue model instead of using the Markov chain of Section 4.1. In such queues, the stationary distribution of the Markov chain is given by a closed formula. Thus, the computation of the distribution is done in a constant time due to the closed formula while the exact computation is much longer. Hence we obtain an approximate solution of the expected cost very quickly. In order to find the best threshold m for which it is better to activate or deactivate, we proceed by comparing approximated costs of having k VMs activated with the one of having $k + 1$ (respectively $k - 1$) VMs activated added by the activation (respectively deactivation) cost.

Design of the heuristic Let us recall, the formulas of the stationary distributions in $M/M/k/B$ queues. Let $\hat{\pi}(k, m)$ be the stationary distribution of state (m, k), we have:

$$\hat{\pi}(k,m) = \hat{\pi}(k,0) \cdot \frac{\rho^m}{m!} \quad \text{for } 1 \leq m \leq k \quad \text{and} \quad \hat{\pi}(k,m) = \hat{\pi}(k,0) \cdot \frac{a^m \cdot k^k}{k!} \quad \text{for } k < m \leq B.$$

where $\rho = \dfrac{\lambda}{\mu}$, $a = \dfrac{\rho}{k} < 1$ and where $\hat{\pi}(k, 0)$ is:

$$\hat{\pi}(k,0) = \left(\sum_{m=0}^{k-1} \frac{\rho^m}{m!} + \frac{k^k}{k!}(B - k + 1) \right)^{-1} \text{ if } a = 1 \text{ and } \hat{\pi}(k,0) = \left(\sum_{m=0}^{k-1} \frac{\rho^m}{m!} + \frac{\rho^k}{k!} \frac{1 - a^{B-k+1}}{1 - a} \right)^{-1} \text{ if } a \neq 1.$$

Let us define $\hat{C}_k(m)$ as the approximated cost of having k VMs turned on and m requests in the system by:

$$\hat{C}_k(m) = \hat{\pi}(k,m)\big(m \cdot C_H + k \cdot C_S\big) + \hat{\pi}(k,B) \cdot \lambda \cdot C_R\,;$$

define $\hat{C}^A_{k+1}(m)$ as the approximated cost of having m requests in level $k+1$ knowing that we have just activated a new VM by:

$$\hat{C}^A_{k+1}(m) = \hat{\pi}(k+1,m)\big(m \cdot C_H + (k+1) \cdot C_S\big) + \hat{\pi}(k+1,B) \cdot \lambda \cdot C_R + \hat{\pi}(k,m-1)\lambda C_A\,.$$

Let us define $\hat{C}^D_k(m)$ as the approximated cost of having m requests in level k knowing that we have just deactivated a new VM by:

$$\hat{C}^D_k(m) = \hat{\pi}(k,m)\,(mC_H + kC_S) + \hat{\pi}(k,B)\lambda C_R + \hat{\pi}(k+1,m+1)(k+1)\mu C_D\,.$$

We want to compare these costs to know whether we should activate, deactivate or let the servers unchanged.

For the activation thresholds. We define $\phi^A_k(m) = \hat{C}_k(m) - \hat{C}^A_{k+1}(m)$. If $\phi^A_k(m) < 0$ then it is better to not activate a new server while if $\phi^A_k(m) \geq 0$ it is better to activate a new one. There is no evidence of the monotonicity of $\phi^A_k(m)$ in m, nevertheless we choose the threshold by $F_k = \min\big\{m : \phi^A_k(m) \geq 0\big\}$. To compute the whole thresholds, all k are studied in an ascending order. For a fixed level k, we need to compute $\phi^A_k(m)$ for all m and stop when the function $\phi^A_k(m)$ is larger than 0.

For deactivation thresholds. We define $\phi^D_k(m) = \hat{C}_{k+1}(m) - \hat{C}^D_k(m)$. If $\phi^D_k(m) < 0$ then it is better to stay with $k+1$ servers while if $\phi^D_k(m) \geq 0$ it is better to deactivate a virtual machine. Similarly, there is no evidence about the monotonicity of $\phi^D_k(m)$ but we define $R_k = \min\big\{m : \phi^D_k(m) \geq 0\big\}$. The computation of the whole deactivation thresholds is similar to the activation ones.

Markov chains approximation methods coupled with local search heuristics The initialisation of the local search heuristics presented in 4.3.1 either was totally random or simply chooses the bounds (lowest values or highest values) of the solution space. The aim here is to improve the speed and the accuracy of these heuristics by coupling them with the *MMKB approximation* used as an initialisation step. This initialisation aims at finding an initial solution that will be in the basin of attraction of the optimal solution. Starting with this solution improves the ability of local search algorithms to reach the best solution in a faster time. This new method behaves better than the initialisation based on *Fluid approximation* adapted from [14] in the work [9]. We define by **Alg MMK** the coupled heuristic with MMKB approximation.

5 Computing policies with Markov Decision Process

We consider here the multi-server queue of Section 3.1 and the controlled model in which only a single virtual resource can be activated or deactivated at a time decision. This optimal control problem is a continuous time process and thus should be solved using a Semi Markov Decision Process (SMDP). In this section, we first describe the uniformised SMDP and its solving. Then we underline the differences between the optimal control model and the optimisation problem of Section 4.3.

5.1 The SMDP model

5.1.1 Elements of the SMDP

The state space is the one defined in Section 3: $\mathcal{S} = \{0,1,\ldots,B\} \times \{1,\ldots,K\}$. Hence, a state $x \in \mathcal{S}$ is such that $x = (m,k)$ where m is the number of customers in the system and k the number of active servers. Similarly, the action space $\mathcal{A} = \{-1,0,1\}$ represents respectively: deactivate one machine, left unchanged the active servers, or activate one machine.

The system evolves in continuous time and at some epoch a transition occurs. When a transition occurs, the controller observes the current state and reacts to adapt the resources by activating or deactivating the virtual machines. The activation or deactivation are instantaneous and then the system evolves until another transition occurs. Two events can occur: an arrival with rate λ which increases the number of customers present in the system by one or a departure which decreases the number of customers by one.

Let $x = (m,k)$ be the state and a the action we define $N(k+a)$ as the real number of active VM after the action a was triggered. We have $N(k+a) = \min\{\max\{1,k+a\},K\}$. It follows that the transition rate is equal to

$$\begin{cases} \lambda & \text{if } y = \big(\min\{m+1,B\},N(k+a)\big) \\ \mu \cdot \min\{m,N(k+a)\} & \text{if } y = \big(\max\{0,m-1\},N(k+a)\big) \end{cases}.$$

5.1.2 Uniformised Transition Probabilities

We apply here the standard method to deal with continuous time MDP: the uniformisation framework. We follow the line of chapter 11 of [26] to define the uniformised MDP.

We define $\Lambda(m,k,a)$ as the rate per state. We have $\Lambda(m,k,a) = \lambda + \mu \cdot \min\{m, N(k+a)\}$. From now on, any component denoted by "$\sim$" refer to the uniformised process. We thus define the uniformisation rate by $\tilde{\Lambda}$ with $\tilde{\Lambda} = \max_{(m,k,a)} \Lambda(m,k,a) = \lambda + K\mu$ which is the maximum transition rate. The transition probability from state x to state y when action a is triggered in the uniformised model is denoted by $\tilde{p}(y|x,a)$.

We have:

$$\tilde{p}(y|x,a) = \begin{cases} \frac{\lambda}{\tilde{\Lambda}} & \text{if } y = (m+1, N(k+a)) \\ \frac{\mu \min\{m, N(k+a)\}}{\tilde{\Lambda}} & \text{if } y = (m-1, N(k+a)) \\ \frac{\tilde{\Lambda} - \Lambda(m,k,a)}{\tilde{\Lambda}} & \text{when } y = (m,k) \\ 0 & \text{otherwise}, \end{cases}$$

when $x = (m,k)$ such that $0 < m < B$ and a is arbitrary ; when $x = (B,k)$ with $a \neq 0$, or when $x = (B,k)$ with $k \neq K$ and $a \neq +1$ or also when $x = (B,k)$ with $k \neq 1$ and $a \neq -1$; when $x = (1,k)$ with $a \neq 0$, or when $x = (1,k)$ with $k \neq K$ and $a \neq +1$, or also when $x = (1,k)$ with $k \neq 1$ and $a \neq -1$. We have:

$$\tilde{p}(y|x,a) = \begin{cases} \frac{\mu \min\{m, N(k+a)\}}{\tilde{\Lambda}} & \text{if } y = (B-1, N(k+a)) \\ \frac{\tilde{\Lambda} - \mu \min\{m, N(k+a)\}}{\tilde{\Lambda}} & \text{when } y = (B,k) \\ 0 & \text{otherwise}, \end{cases}$$

when $x = (B,k)$ with $a = 0$, or when $x = (B,k)$ with $k \neq K$ and $a \neq +1$, or also when $x = (B,k)$ with $k \neq 1$ and $a \neq -1$. We have

$$\tilde{p}(y|x,a) = \begin{cases} \frac{\lambda}{\tilde{\Lambda}} & \text{if } y = (1, N(k+a)) \\ \frac{\tilde{\Lambda} - \lambda}{\tilde{\Lambda}} & \text{when } y = (0,k) \\ 0 & \text{otherwise}, \end{cases}$$

when $x = (0,k)$ with $a = 0$, or when $x = (0,k)$ with $k \neq K$ and $a \neq +1$, or when $x = (0,k)$ with $k \neq 1$ and $a \neq -1$.

5.1.3 The uniformised stage costs

We take the definition of costs of Section 3.1. Instantaneous costs are charged only once and are related to activation an deactivation and losses. Accumulated costs are accumulated over time and are related to consumption and holding cost. After uniformisation of instantaneous and accumulated costs (Chapter 11.5 of [26]), we obtain the following equation for $x = (m,k)$:

$$\tilde{c}(x,a) = \big(C_A \mathbb{1}_{\{a=1\}} + C_D \mathbb{1}_{\{a=-1\}}\big) \frac{\Lambda(x,a)}{\tilde{\Lambda}} + \frac{\lambda}{\tilde{\Lambda}} C_R \mathbb{1}_{\{m=B\}} + \frac{1}{\tilde{\Lambda}} \big(N(k+a) \cdot C_S + m \cdot C_H\big).$$

Objective function The objective function defined in Equation (1) translates for all $x \in \mathcal{S}$ into

$$\rho^\pi(x) = \lim_{N \to \infty} E^\pi \Big[\frac{1}{N} \sum_{t=0}^{N-1} \tilde{c}(x_t, \pi(x_t)) \mid x_0 = x \Big],$$

for a given policy π. The value ρ^π is the expected stage cost and equals $\bar{C}$ when actions follow policy π.

5.2 Solving the MDP

5.2.1 Classification of the SMDP

Our SMDP is an average cost model. This is why, the expected stage costs depend on the recurrent properties of the underlying Markov chain that is generated by a deterministic policy. A classification is then necessary to study them. We use the classification scheme of [26].

Definition 6 (Chapter 8.3.1 in [26])**.** *A MDP is:*

i) Unichain *if, the transition matrix corresponding to every deterministic stationary policy is unichain, that is, it consists of a single recurrent class, plus a possibly empty set of transient states;*

ii) Multichain *if, the transition matrix corresponding to at least one deterministic stationary policy contains two or more recurrent classes;*

iii) Communicating *if, for every pair of state x and y in $\mathcal{S}$, there exists a deterministic stationary policy π under which y is accessible from x, that is, $p_\pi^n(y,x) > 0$ for some $n \geq 1$;*

Proposition 2. *The MDP is multichain. There is a stationary deterministic policy with monotone hysteresis properties that induces a corresponding Markov chain with more than two different recurrent classes.*

Proof. We assume that $K \geq 2$ and let k be such that $k \in [1, \dots, K]$. We define the policy q as follows. For any level l with $l < k-1$, we have only activation. There exists $m \in [0, \cdots, B]$ such that $q(m', l) = 1$ for any $m \leq m'$ and $q(m,l) \leq q(m, l-1)$. For level k we have neither activation nor deactivation and $q(m,k) = 0$ for all $m \in [0, \cdots, B]$. For any level l with $l > k+1$, we have only deactivation. There exists $m \in [0, \cdots, B]$ such that $q(m', l) = -1$ for any $m' \leq m$ and $q(m,l) \geq q(m, l+1)$. Therefore, we have three recurrent classes: the level k, the level $k-1$ and the level $k+1$ (and two recurrent classes for $K = 2$). See supplementary materials for details. □

Lemma 3. *The MDP is communicating. There exists a stationary isotone hysteresis policy such that the corresponding Markov chain is irreducible.*

Proof. We exhibit such a policy. Let π be the deterministic stationary policy such that the thresholds l_k are defined by $l_k = 0$ and the thresholds L_k by $L_k = B$ for all k. The induced Markov chain is irreducible since any level can be reached from another one (when $m = 0$ or $m = B$) and since in a given level all the states are reachable from any state. Thus, we have that $p_\pi^n(y,x) > 0$ for some $n \geq 1$ for all couples (x,y). Therefore the MDP is communicating. □

Bellman Equations In the *multichain* case, the Bellman Equations are composed by two equations. In the uniformised model, we then have ([26]) the two following optimality equations:

$$\min_{a \in A} \left\{ \sum_{y \in \mathcal{X}} \tilde{p}(y \,|\, x, a)\rho(y) - \rho(x) \right\} = 0 \;\text{ and }\; U(x) = \min_{a \in B_x} \left\{ \tilde{c}(x,a) - \frac{\rho(x)}{\tilde{\Lambda}} + \sum_{y \in \mathcal{X}} \tilde{p}(y \,|\, x, a) \cdot U(y) \right\}$$

for all $x \in \mathcal{X}$, where

$$B_x = \left\{ a \in \mathcal{A} \;\middle|\; \sum_{y \in \mathcal{X}} \tilde{p}(y \,|\, x, a)\rho(y) \;=\; \rho(x) \right\}.$$

It could be noticed that in the unichain case, $B_x = \mathcal{A}$ and that the two equations reduce to only the second one. These two non linear equation systems should be numerically solved to find $U(x)$ and $\rho(x)$. Once these terms are approximated we deduce the optimal policy with:

$$q(x) = \arg\min_{a \in B_x} \left\{ \tilde{c}(x,a) + \sum_{y \in \mathcal{X}} \tilde{p}(y \,|\, x, a) \cdot U(y) \right\}.$$

5.2.2 Algorithms

We describe here the choice of the algorithms used to solve this multichain and communicating model. Computing multichain model is much complicated namely since testing if an induced Markov chain is unichain is a NP complete problem. We first show that we actually can use some unichain algorithms due to the communicating properties of our SMDP, then we present two structured algorithms based on hysteresis properties.

Unichain algorithms From [26] it exists a *multichain policy iteration algorithm* that solves multichain models. It requires to solve two equation systems and thus is time consuming. However, by Proposition 3, our MDP is communicating and in Theorem 8.3.2 of [26] it is proved that unichain value iteration algorithm converges to the optimal value in communicating models. This property allows us to use the unichain value iteration algorithm.

There is also, in [26], a policy iteration algorithm for multichain models. It requires to start with an initial unichain policy and its inner loop roughly differs from the unichain case in order to keep some properties. We decide to use here algorithms based on unichain policy iteration. There does not exist theoretical guarantee of their convergence, but we showed in numerical experiments that they always converge to the same policy than value iteration.

Four different usual unichain algorithms will be considered: *Value Iteration*, *Relative value iteration*, *Policy Iteration modified* and *Policy Iteration modified adapted*, which adapts its precision in the policy evaluation step. They are respectively referred by *VI*, *RVI*, *PI* and *PI Adapt*. The algorithms are all described in [26] and are already implemented in the software [28].

Structured Policies algorithm We now integrate hysteresis properties in the algorithms. Two classes of policies have been investigated: Double Level class (Definition 1) and Monotone Hysteresis class (Definition 4). The goal is to plug hysteresis assumptions during the policy improvement step of policy iteration (Policy Iteration has two major steps). This allows to test less actions at each iteration and to speed up the algorithm. Two algorithms are implemented: one for the double level properties (referred as *DL-PI*) and one for the hysteresis properties (referred as *Hy-PI*).

On the other hand, we do not have theoretical guarantee that these methods converge, first because we do not theoretically know if hysteresis policies are optimal for our model, and second because the underlying PI also has no convergence guarantees in multichain. Nevertheless, in the numerical experiments we made (see later), all the optimal policies returned by classical algorithms have hysteresis properties and furthermore all MDP algorithms considered here (structured as well as classical) returned the same solution. This therefore underlines the interest of considering such hysteresis policies especially since the gain in running time is obvious as observed in the experiments.

Computation of the hysteresis thresholds The non structured MDPs (also called simple MDPs) do not assume any restrictions for their policy research. Thus, they return the optimal policy and so return a decision rule q^* which gives the optimal action to take but not the thresholds. Therefore, we need to test the hysteresis property and to compute the l and L hysteresis thresholds consistently with Definition 4. Let q^* be the optimal policy returned by the PI algorithm. We check if q^* is monotone, if not the optimal policy is not hysteresis. If so, we proceed as follows. We consider for all k the set $\{m \mid q^*(m,k) = 1 \text{ and } q^*(m-1,k) = 0\}$. If the set is of size 2 then the policy is not hysteresis, if the set is of size 1 then $L_{k+1} = m$ and if the set is empty then $L_{k+1} = \infty$. Also, we consider for all k the set $\{m \mid q^*(m+1,k) = 0 \text{ and } q^*(m,k) = -1\}$. If the set is of size 2 then the policy is not hysteresis, if the set is of size 1 then $l_{k+1} = m$ and if the set is empty then $l_{k+1} = 0$.

5.3 Theoretical comparison between the two approaches MC and MDP

Although they seem very similar these two models present some rather subtle theoretical differences with notable consequences. They are studied here while the numerical comparisons will be carried out in Section 7.

5.3.1 Number of activated resources

A major difference between the two approaches is related to the class of policy they consider and the consequences on the management of resources. The (MC) approach (Section 4), deals with *strictly isotone* policies (*i.e.* $0 = l_1 < l_2 < \ldots < l_K \leq B$ and $0 = L_1 < L_2 < \ldots < L_K \leq B$). Therefore, L_K can not be infinite, and in this model all the K servers should be activated. Furthermore, since inequalities are strict $l_K \geq K$, the only state in which there is only one active server is restricted to the level $k = 1$. Strictly isotone assumption is required to keep the size of the chain constant (see [22]). On the other hand, the (MDP) approach either does not assume any structural property on the policy or consider isotone policy (see Definition 3). In this approach constraints are looser and it is possible not to have all the servers activated and conversely to have many resources activated even if the queue is empty.

The resource managements follow the optimal policies returned by each of the approaches. Three categories of resource managements have been identified. We noticed that these categories depend on the relative values of the parameters (λ, μ) between them. However, we do not know how to precisely quantify their borders.

Definition 7. *These categories are:*

1. Medium arrival case*: All VMs are turned On and turned Off in both approaches. This occurs when the system load is medium, i.e. the arrival rate λ is close to the service rate $k * \mu$.*

2. Low arrival case*: All VMs are turned On and turned Off in (MC) while in (MDP) some VMs will never be activated. This occurs when the system load is low, i.e. the arrival rate λ is very small compared to the service rate $k * \mu$.*

3. High arrival case*: All VMs are turned On and turned Off in (MC) while in (MDP) some VMs will never be deactivated. This occurs when the system load is in high, i.e. the arrival rate λ is very large compared to the service rate $k * \mu$.*

Proposition 3. *The above classification helps us to claim that strictly isotone policies are not optimal and that MDP approach performs better.*

Proof. During the numerical experiments in Section 7 we identify numerous cases in which strictly isotone policies are not optimal (see supplementary materials for details). This is mainly due to the phenomenon of non activation or non deactivation exposed above: examples of non-optimality are found in both low arrival and high arrival categories. The structured (MDP)s have less constraints and return the optimal solution which is the one computed by simple MDPs. □

The proposition above highlights another benefit of MDP approaches since they allow to size the exact number of machines to be activated (this is particularly true in the *low arrival* case). Hence, if in a policy resulting from a MDP there exits a k such that for all $l > k$ and for all $m \in \{1, \ldots, B\}$ we have $q(m, l) < 1$, then $K - k$ machines are not necessary. This is not true in MC heuristics, nevertheless, it is possible to adapt the previous heuristics to find the optimal number of VM to be activated in *low arrival* case. This requires running the algorithms for each level k with $k \in \{1, \ldots, K\}$. Then take the level k which has the smallest cost says k^*. If $k^* = K$ all machines must be turned on in K, otherwise only k^* machines are important and the other ones useless. But this method is time consuming. Also, such an adaptation for *high arrival* cases is not obvious.

5.3.2 Different temporal sequence of transitions

The last point to investigate is the difference in dynamical behaviour between transitions. This difference is slight and has no effect on average costs, nevertheless it induces a difference on the values of the thresholds.

Actually, in this system, there are two kinds of transitions: *natural transitions* which are due to events (departures or arrivals) and *triggered transitions* which are caused by the operator (activation or deactivation). These two transitions are instantaneous. Due to their intrinsic definitions, the models considered here do not observe the system at the same epochs. Hence, in the Markov chain the system is observed just before a natural transition while in the MDP the system is observed just after a natural transition. More formally, let us assume that x is the state before a natural transition: it is seen by the Markov chain. Then the transition occurs and the state changes instantaneously in x' that is the state seen by the MDP. The controller reacts and the triggered transition occurs, thus the system moves in x'' in which the system remains until the next event which will cause the next natural transition. Since state changes are instantaneous, this has no impact on costs. See supplementary materials for details.

In the *medium case arrival* defined by Definition 7, thresholds are fully comparable and we have the following lemma

Lemma 4. *In the medium case arrival, hysteresis thresholds of Markov chain and SMDP hysteresis threshold can be inferred from each other. Let $(F_1, \ldots, F_{K-1})$ and $(R_1, \ldots, R_{K-1})$ be the thresholds of the MC model and let $(L_2, \ldots, L_K)$ and $(l_2, \ldots, l_K)$ be the SMDP thresholds. Then, for all $k \in \{1, \ldots, K-1\}$, we have: $L_{k+1} - 1 = F_k$ and $l_{k+1} + 1 = R_k$.*

Proof. We make the proof for an activation threshold. Let assume that the state just before a transition is $x = (F_k, k)$, if an arrival event occurs then the system moves instantaneously in a state $x' = (F_k + 1, k)$. According Definition 5, a virtual machine is activated and then the system instantaneously moves again in $x'' = (F_k + 1, k + 1)$. Observe now, that the state x' (according Definition 2) is the state in which the SMDP observes the system an takes it activation decision. Thus we have $L_{k+1} = F_k + 1$. The proof works similarly for deactivation. □

6 Real model for a Cloud provider

We want to apply the algorithms presented before on concrete models derived from real environment values. We now consider models with real data for energy consumption, financial costs of virtual machines, and financial costs of Service Level Agreement (SLA). However, one of the difficulties of such an approach lies in the small number of works that consider both energy and quality of service. Hence, we should take different separate elements to build such a model. This allows us to assess the real impact, especially the financial cost and the reduction of the energy

consumption, that a cloud owner can generate with such algorithms and to determine the optimisation method it should use.

6.1 Cost-Aware Model with Real Energy Consumption and Pricing

Energy Consumption Data The work [4] details information about real energy consumption in virtualised system. The authors have made some simulations on a real datacenter, *grid5000*, and have recovered data about energy consumption for VMs based on some physical servers whom characteristics will be described in Section 7.3. This will allow us to take these real values in Watt for energy consumption of virtual machines, when we built a concrete case.

Financial Cost Recall that the model proposed here is *cost-aware* and evaluates a financial cost for the cloud owner and that our goal is to minimise the operational cost while guaranteeing the QoS. One way to build real experiments is to consider real data about energy consumption and to transform Watts consumption in a financial cost considering the price in euros of the KWh in France defined by the national company. So, once we have our energy costs fixed depending on the system settings (CPU, vCPU, etc.), we can translate these Watt costs into financial costs.

Service Level Agreement In order to capture the realistic scheme of the Service Level Agreement, we modify our cost function by changing the performance part. Therefore the mean global cost for the cloud provider is now:

$$\begin{aligned}\overline{C}(m,k) = C_p \cdot (m - N_{SLA}, 0)^+ + C_S \cdot k + C_p \cdot \lambda \cdot \mathbb{1}_{\{m=B, k=K\}} + C_A \cdot \lambda \cdot \mathbb{1}_{\{m=F_k,\, k<K\}} \\ + \; C_D \cdot \mu \cdot \min\{m,k\} \cdot \mathbb{1}_{\{m=R_{k-1}+1,\; 2\leq k\leq K\}} + C_{static}\,,\end{aligned}$$

where C_{static} represents the static financial cost obtained from the idle energy consumption in Watt of the physical server hosting the virtual machines. The energy part of the cost function remains the same with the activation, deactivation, and energy cost of a VM being used. For the performance part, the issue we had was to give a meaning and real values for the holding cost since this one does not translate directly from usual SLA models. Nevertheless, the holding cost can represent the penalty to pay if the QoS is not satisfied. Hence, usually SLA contracts are defining penalty costs when the mean response time exceeds a pre-defined threshold T_{SLA} corresponding to a maximal time to process the jobs. Above this threshold, the cloud owner will have to pay a penalty C_p which in our setting will be the price the customer pays for one hour of service of a virtual machine [29] (full reimbursement). Let R_T be the mean response time, we thus want that: $E[R_T] \leq T_{SLA}$.

We can see this mean response time in terms of number of jobs waiting for in the system, i.e. reducing the mean response time in the system is equivalent to reduce the mean number of jobs in the system. Hence, if $E(N)$ is the mean number of customers, then from Little's law, we have: $E[N] \leq T_{SLA} \cdot \lambda$ and one may define a customer threshold as $N_{SLA} = T_{SLA} \cdot \lambda$ which is the maximum number of jobs, that the system should have to satisfy the required *QoS*. The cloud owner will therefore pay an holding cost penalty of C_p by customer each time $m > N_{SLA}$. This cost definition is meaningful since the cloud owner will have to turn on sufficient number of VMs to satisfy the *QoS*, still minimising its operational costs.

6.2 Real Packets and CPU utilisation

After defining some real financial costs for energy and performance, we may want to give a meaning and real values to λ and μ. We consider in our model one request arriving at a time and one VM is processing one request at a time. Describing these parameters in this way, allows us to take some concrete values that can happen in real schemes. Looking at workload traces from MetaCentrum Czech National Grid data [30], we can take a number of requests per hour arriving in a real data center. We also have some experimental results about mean run-time of jobs, which give us real values of μ. Let us refer to 7.3 for numerical results with real scenarios.

7 Numerical experiments

This part is devoted to numerical experiments and the comparison between all previous algorithms. We have implemented all methods in C++, with the stand alone library [28].

7.1 Experiments design and results

All the experiments were launched with a processor Intel Xeon X7460 @2,66 GHz with 6x4 cores (24 threads) and 16 GB of RAM. We display in Table 1 the comparisons in terms of accuracy and time execution for (MC) heuristics and (MDP) algorithms. Numerical experiments have been done for different Cloud model parameters. The parameters

of the system have been taken arbitrarily in order to rank the different algorithms by comparing their accuracy and execution time. We defined four scenarios with different scales: **Scenario A**: K=3, B=20; **Scenario B**: K=5, B=40; **Scenario C**: K=8, B=60; **Scenario D**: K=16, B=100. We call *Instance* a set of parameters $\lambda, \mu, C_a, C_d, C_h, C_s, C_r$. The algorithms have been tested on 46656 instances. Costs (C_a, C_d, C_h, C_s) were taking values in $[0.5, 1, 2, 5, 10, 20]$, C_r in $[1, 10, 100, 1000, 5000, 10000]$, queueing parameters λ, μ in $[0.5, 1, 2, 5, 10, 20]$. We first studied numerical experiments in the (MC) approach by comparing accuracy and time execution between different heuristics. For Scenario A, accuracy is computed based on optimal solution generated by exhaustive search. For higher scale scenarios (B,C,D), we compute accuracy regarding the best solution among the heuristics. Then, we compared all the (MDP) algorithms with the value iteration solution (convergence is known from Section 5.2.2) before comparison with the heuristics to compare both approaches. We finally ran the experiments for a concrete cloud scenario with real data, showing how we can effectively benefit from the best methods to calculate optimal thresholds.

Models	**Algorithms**	**Scenario A**		**Scenario B**		**Scenario C**		**Scenario D**	
		Time (sec)	% Opt	Time (sec)	% Min	Time (sec)	% Min	Time (sec)	% Min
MC Heuristics	**BPL**	0,063 sec	96,9 %	2,684 sec	86,81%	37,23 sec	79,57 %	757 sec	55,39%
	BPL Agg	0,047 sec	96,9 %	1,163 sec	86,81%	10,27 sec	79,57 %	186 sec	55,39%
	BPL-MMK Agg	0,036 sec	97,78 %	0,53 sec	96,59 %	5,16 sec	95,39 %	31,7 sec	87,24 %
	NLS	0.043 sec	92,70 %	1,007 sec	62,24 %	12,38 sec	43,51 %	257 sec	20,44 %
	NLS Agg	0,034 sec	92,70 %	0,458 sec	62,24 %	4,05 sec	43,51 %	61 sec	20,44 %
	NLS-MMK Agg	0,014 sec	96,32 %	0,06 sec	93,58 %	0,9 sec	89,54 %	8,39 sec	76,17 %
		Time (sec)	% Opt	Time (sec)	% Opt	Time (sec)	% Opt	Time (sec)	% Opt
MDP	**VI**	0.0057 sec	100 %	0.021 sec	100 %	0.0406 sec	100 %	0,0944 sec	100 %
	RVI	0.0057 sec	100 %	0.021 sec	100 %	0.0406 sec	100 %	0,095 sec	100 %
	PI	0.0028 sec	100 %	0.0124 sec	100 %	0.0214 sec	100 %	0.0606 sec	100 %
	PI Adapted	0,0023 sec	100 %	0,0117 sec	100 %	0,0201 sec	100 %	0,0583 sec	100 %
	DL-PI	0,00115 sec	100 %	0,0072 sec	100 %	0,0105 sec	100 %	0,0452 sec	100 %
	Hy-PI	0,00113 sec	100 %	0,0069 sec	100 %	0,0100 sec	100 %	0,0442 sec	100 %
		Time (sec)	% Opt	Time (sec)	% Opt	Time (sec)	% Opt	Time (sec)	% Opt
Comparison	**NLS-MMK Agg**	0,014 sec	64,82 %	0,06 sec	61,52 %	0,9 sec	59,82 %	8,39 sec	52,61 %
	BPL-MMK Agg	0,036 sec	65,15 %	0,53 sec	62,65 %	5,16 sec	61,69 %	31,7 sec	57,1 %
	DL-PI	0,00115 sec	100 %	0,0072 sec	100 %	0,0105 sec	100 %	0,0452 sec	100 %
	Hy-PI	0,00113 sec	100 %	0,0069 sec	100 %	0,0100 sec	100 %	0,0442 sec	100 %

Table 1: Numerical experiments for comparison of heuristics and MDP algorithm

7.2 Experiments analysis

7.2.1 Assessing heuristics approaches

We want to assess the selected heuristics of [9] and the gain of the improvements we propose.

Aggregation strongly improves algorithms time speed We first observe a significant time saving offered by the aggregation method for the same efficiency. The improvement in Scenario A is small, as expected, but the gain increases as the size of the system rises. Hence the running time is divided by about 1.3 when $K = 3$ and $B = 20$ and by about 3.8 when $K = 16$ and $B = 100$ (e.g. BPL algorithm passes from 757 seconds to 186 seconds).

Impact of the M/M/k/B approximation Coupled with heuristics **BPL** and **NLS**, M/M/k/B approximation brings significant improvement considering the efficiency and the time execution. For a large scale case (Scenario D in Table 1), the *BPL MMK Agg* provides the best solution for 87.24 % of the instances in 31.7 seconds, which is the best heuristic in terms of time-accuracy ratio. We can notice in all cases that coupled heuristic always have better accuracy than the heuristic alone. Moreover, the initialisation technique does not only improve efficiency but also time execution. For example in Scenario C, **BPL Agg** has a mean time of 10,27 seconds for 79,57 % accuracy while **BPL-MMk Agg** obtains 95,39 % accuracy in 5,16 seconds, providing a double gain.

7.2.2 Comparison between MDP algorithms

By Section 5.2.2, we theoretically know that value iteration algorithm converges to the exact solution. We observe in these numerical experiments, that all MDP algorithms obtain 100% of accuracy (optimal solution being given by value iteration) in all scenarios. Thus, all MDP algorithms converge to the optimal solution. Concerning the running time, we observe that policy iteration algorithms **PI** and **PI Adapted** are twice as good than value iteration on all studied scenarios. Furthermore, we see that the execution time of the structured MDP algorithms **DL-PI** and **Hy-PI** is divided by 2 compared to **PI Adapted**. This leaves us to conclude that integration of policy with structural properties strongly accelerates the convergence.

7.2.3 Comparison between (MDP) and (MC) approaches

We now compare which approach performs best to obtain optimal thresholds. We first observe that the accuracy difference is striking, indeed, the MC approach is optimal only about half the time. Indeed, from Section 5.3.1, we know that depending on the queueing parameters, (MDP) approach could not activate or deactivate certain virtual resources whereas the (MC) approach always find thresholds for all levels due to constraints on the searched policy. This is why (MC) heuristics are outperformed by (MDP) algorithms. Hence for many instances their optimal cost is higher due to additional activation or deactivation thresholds, generating extra costs. We see that MDP algorithms are faster than heuristics in the four scenarios. Hence for small scenarios **Hy-PI** is about 100 times faster than the best heuristic and about 200 times faster for large scenarios. Therefore, the MDP approach is significantly better than the (MC) approach.

7.3 Numerical experiments for concrete scenarios

Data and scenarios We give now numerical values to our concrete model defined in Section 6 to draw several case-studies and some performance and metrics curves. We consider an hour time unit. We consider from [10] the following physical server: a Taurus Dell PowerEdge R720 hardware with processor Intel Xeon e5-2630 (2.3 GHz) with 6x2 cores (12 threads=logical cores) which hosts the virtual machines with an hypervisor. We consider one core to be one vCPU and we distribute them equally among the virtual machines: 3 VMs with 4 cores per VM, 6 VMs with 2 cores per VM and 12 VMs with 1 core per VM. We have drawn these three different cases where we can use the price of Amazon EC2 instances. From [10], we take the static energy consumption of the physical server around 100W and the dynamic energy consumption of the virtual machines are stated according to the number of cores per VM. Depending on the characteristics of the virtual machines hosted on the physical server, the Watt consumption for activation, deactivation and utilisation is different. For example, one VM with 4 vCPU executing a request needs 40W to work and its activation and deactivation requires a power of 10W. Then, all the energy consumption values are translated in financial costs.

In [30], is described some workloads based on real traces from the MetaCentrum Czech national grid. Since we fixed the hour as our time unit, we draw as an example a number of requests per hour $\lambda = 50$ and number of served requests per hour as $\mu = 5, 10$ or 20 which can be concrete daily scenarios for huge size requests. Table 2 reminds all the values of the parameters of each model.

Parameter	Model A	Model B	Model C
$\#VMs$	3	6	12
Instances	a1.xlarge	t3.small	t2.micro
$\#vCPUs$	4	2	1
RAM(Go)	8	2	1
C_p	0.0914€/h	0.0211€/h	0.0118€/h
C_S	0.00632€/h	0.00316€/h	0.00158€/h
$C_A = C_D$	0.00158€	0.00079€	0.00032€
C_{Static}	0.0158€/h	0.0158€/h	0.0158€/h
B	100	100	100
λ	50 req/h	50 req/h	50 req/h
μ	20 req/h	10 req/h	5 req/h

Table 2: Table of Values for different scenarios

We provide the mean run time of the best algorithm (**Hy-PI**) among concrete experiments: 0,00127 s for Model A ; 0,0073 sec for Model B and 0,0372 sec for Model C. The optimal solution is computed in a short-time and therefore can allow to recompute new policy when the demand is evolving, in order to dynamically adapt to the need.

Mean global costs given SLA parameter or arrival rate We first remark from numerical simulations that the cost decreases when the SLA is less rigid for a fixed arrival rate λ. If the response time fixed by the SLA increases, then the operational costs for the cloud providers decrease since they will pay less penalties to customers and less electricity bill for VMs execution. Secondly, we observe that Model C is always better than Model B and Model A for any possible values of N_{SLA} and for any values of λ given a fixed SLA parameter. We can conclude that it is always better to decompose physical machine in several virtual resources, since it allows a more efficient resource allocation in dynamic scenario. Indeed, the more virtual resources you have, the less you will pay per utilisation since they require less CPU consumption. Moreover, it allows a more flexible system where you can easily adapt virtual resources to the demand.

Thresholds given SLA parameter or arrival rate We also observe that Model C always activates a second virtual resource before the two other models B and A, for any fixed values of λ and SLA parameter N_{SLA}. Furthermore, when

the arrival rate λ is fixed, then all models will activate later when the SLA is less restrictive. Lastly, when the SLA parameter is fixed, we notice that the first activation threshold decreases when λ increases. Indeed, if the demand is growing, it requires more resources and requires faster activation of virtual resources.

Cost evaluation We display an example ($K = 8, B = 60$) showing how the financial cost per hour evolves depending on the given N_{SLA}, or on the load. We can see in Figure 3 the impacts of such parameters on regarding performance and energy costs.

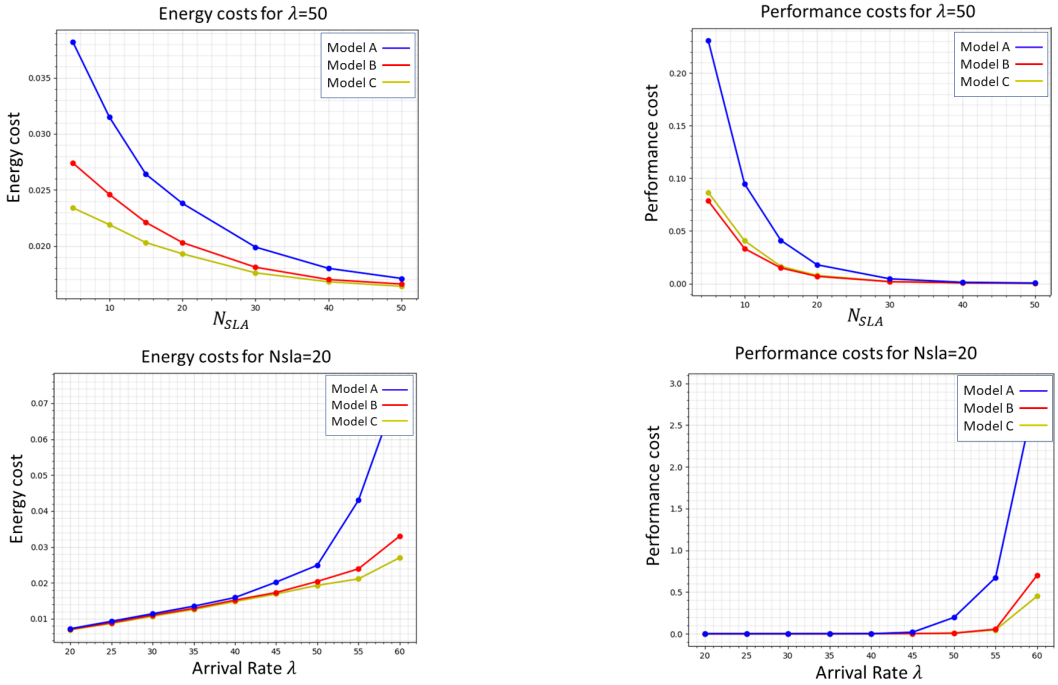

Figure 3: Energy and performance costs given fixed SLA or arrival rate

8 Conclusions and further work

In this paper we have compared theoretically and numerically two different approaches to minimise the global cost integrating both performance and energy consumption in an auto scaling cloud system. The relevance of this study for a cloud provider is to provide an auto scaling policy very quickly so as to minimise its financial cost (around 38 seconds for 128 VMs and 800 customers). We exhibit that the best heuristics we propose is strongly outperformed by SMDP models and that hysteresis present a significant improvement. Our next steps will be to build a cloud prototype to assess the efficiency of our policies in a real context and second to enlarge our model by including more general arrival process as bursty traffic predicted from a measurement tool.

References

[1] Observatoire de l'industrie Electrique. *Le cloud, les data centers et l'énergie*, January 2017.

[2] T. Mastelic and I. Brandic. Recent trends in energy-efficient cloud computing. *IEEE Cloud Computing*, 2:40–47, 2015.

[3] Amazon. AWS Auto Scaling, 2018.

[4] A. Benoit, L. Lefèvre, A.-C. Orgerie, and I. Rais. Reducing the energy consumption of large scale computing systems through combined shutdown policies with multiple constraints. *Int. J. High Perform. Comput. Appl.*, 32(1):176–188, 2018.

[5] S. Shorgin, A. Pechinkin, K. Samouylov, Y. Gaidamaka, I. Gudkova, and E. Sopin. Threshold-based queuing system for performance analysis of cloud computing system with dynamic scaling. In *AIP Conference*, volume 1648, 2015.

[6] M.Y. Kitaev and R.F. Serfozo. M/M/1 queues with switching costs and hysteretic optimal control. *Operations Research*, 47:310–312, 1999.

[7] J. C. S. Lui and L. Golubchik. Stochastic complement analysis of multi-server threshold queues with hysteresis. *Performance Evaluation*, 35:19–48, 1999.

[8] N.M. Asghari, M. Mandjes, and A. Walid. Energy-efficient scheduling in multi-core servers. *Computer Networks*, 59:33–43, 2014.

[9] T. Tournaire, H. Castel, E. Hyon, and T. Hoche. Generating optimal thresholds in a hysteresis queue: a cloud application. *IEEE Mascots*, 2019.

[10] M. Kurpicz, A.-C. Orgerie, and A. Sobe. How much does a VM cost? energy-proportional accounting in VM-based environments. In *PDP*, pages 651–658, 2016.

[11] J. Krzywda, A. Ali-Eldin, T.E. Carlson, P.-O. Ostberg, and E. Elmroth. Power-performance tradeoffs in data center servers: DVFS, CPU pinning, horizontal and vertical scaling. *Future Generation Computer Systems*, pages 114–128, 2018.

[12] I.J.B.F. Adan, V.G. Kulkarni, and A.C.C. van Wijk. Optimal control of a server farm. *INFOR: Information Systems and Operational Research*, 51(4):241–252, 2013.

[13] A. Gandi, M. Harchol-Balter, and I. Adan. Server farms with setup costs. *Performance Evaluation*, 67(11):1123–1138, 2010.

[14] I. Mitrani. Managing performance and power consumption in a server farm. *Annals of Operations Research*, 202:121–134, 2013.

[15] J. R. Artalejo, A. Economou, and M. J. Lopez-Herrero. Analysis of a multiserver queue with setup times. *Queueing Systems*, 51(1-2):53–76, 2005.

[16] D. Ardagna, G. Casale, M. Ciavotta, J.F. Pérez, and W. Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *JISA*, 5, 2014.

[17] J. Teghem. Control of the service process in a queueing system. *EJOR*, 23(2):141–158, 1986.

[18] Z. Yang, M.-H. Chen, Z. Niu, and D. Huang. An optimal hysteretic control policy for energy saving in cloud computing. In *GLOBECOM*, pages 1–5, 2011.

[19] N. Lee and V. G. Kulkarni. Optimal arrival rate and service rate control of multi-server queues. *Queueing Syst. Theory Appl.*, 76(1):37–50, 2014.

[20] D.S. Szarkowicz and T.W. Knowles. Optimal control of an M/M/S queueing system. *Operations Research*, 33(3):644–660, 1985.

[21] D.-P. Song. *Optimal Control and Optimization of Stochastic Supply Chain Systems*. Springer-Verlag, 2013.

[22] O.C. Ibe and J. Keilson. Multi-server threshold queues with hysteresis. *Performance Evaluation*, 21:185–213, 1995.

[23] M.M. Kandi, F. Aït-Salaht, H. Castel-Taleb, and E. Hyon. Analysis of performance and energy consumption in the cloud. In *EPEW*, pages 199–213, 2017.

[24] A.A. Kranenburg and G.J. van Houtum. Cost optimization in the (S-1, S) lost sales inventory model with multiple demand classes. *Oper. Res. Lett.*, 35(4):493–502, 2007.

[25] D. Efrosinin, I. Gudkova, and N. Stepanova. Performance analysis and optimal control for queueing system with a reserve unreliable server pool. In *ITMM*, pages 109–120, 2019.

[26] M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[27] S.K. Hipp and U.D. Holzbaur. Decision processes with monotone hysteretic policies. *Operations Research*, 36(4):585–588, 1988.

[28] A. Jean-Marie. marmoteCore: a Markov modeling platform. In *VALUETOOLS*, 2017.

[29] Amazon. Amazon EC2 pricing, 2019.

[30] D. Guyon, A-C. Orgerie, C. Morin, and D. Agarwal. Involving users in energy conservation: A case study in scientific clouds. *International Journal of Grid and Utility Computing*, 2018.

A Numerical experiments example

Example 2. *We display an example for all cases according to Definition 5.1, to illustrate the difference of optimal solutions between both approaches. Solutions returned by MDP algorithms are displayed without the shift operation. We take the following costs values* $C_a = C_d = 2,\ C_h = C_s = 5,\ C_r = 10$ *with* $K = 16,\ B = 100$ *scenario.*

Medium arrival case *Consider the following queueing parameters:* $\lambda = 500,\ \mu = 100$.

The MDP algorithm provides the following solution: $C^* = 25,239$*; and*

$L^* = [2, 4, 5, 7, 8, 10, 11, 13, 15, 16, 18, 19, 21, 23, 24]$;

$l^* = [0, 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$.

The heuristic in the (MC) model provides the following solution: $C^* = 25,239$*; and :*

$F^* = [1, 3, 4, 6, 7, 9, 10, 12, 14, 15, 17, 18, 20, 22, 23]$;

$R^* = [1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]$.

We can observe here that the two solutions are identical since (MDP) algorithm finds thresholds in all levels.

Low arrival case *Consider the following queueing parameters:* $\lambda = 50,\ \mu = 100$.

The MDP algorithm provides the following solution: $C^* = 9,99323$*; and :*

$L^* = [12, 26, 39, 52, 64, 77, 90, \infty, \infty, \infty, \infty, \infty, \infty, \infty, \infty]$;

$l^* = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]$.

The heuristic in the (MC) model provides the following solution: $C^* = 9,99375$*; and :*

$F^* = [11, 25, 28, 39, 51, 58, 70, 72, 88, 94, 95, 96, 97, 98, 99]$;

$R^* = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]$.

Since the load in the system is very low, the (MDP) algorithm does not need to turn ON all virtual resources. However, we observe that (MC) approach still activate all levels but tries to activate as late as possible to compensate that it does not need to turn ON more virtual resources. It induces a small supplementary costs since these activations will 'never' happen in this system.

High arrival case *Consider the following queueing parameters:* $\lambda = 1000,\ \mu = 100$.

The MDP algorithm provides the following solution: $C^* = 119,756$*; and :*

$L^* = [2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 16, 24, 56, 69, 82]$;

$l^* = [0, 0, 0, \ldots, 0, 0, 0, 10, 12, 14]$.

The heuristic in the (MC) model provides the following solution: $C^* = 220,879$*; and :*

$F^* = [2, 4, 6, 8, 11, 13, 16, 21, 27, 37, 48, 61, 75, 88, 96]$;

$R^* = [1, 2, 3, 4, 5, 6, 7, 8, 11, 16, 22, 26, 29, 30, 34]$.

We can observe here that (MC) approach still define deactivation thresholds in any levels (due to its model and isotone hysteresis constraints) leading to a higher mean cost.

B Figures

B.1 Multichain MDP example

We show a scenario where the MDP policy q will generate a multichain Markov chain where we can observe several recurrent classes.

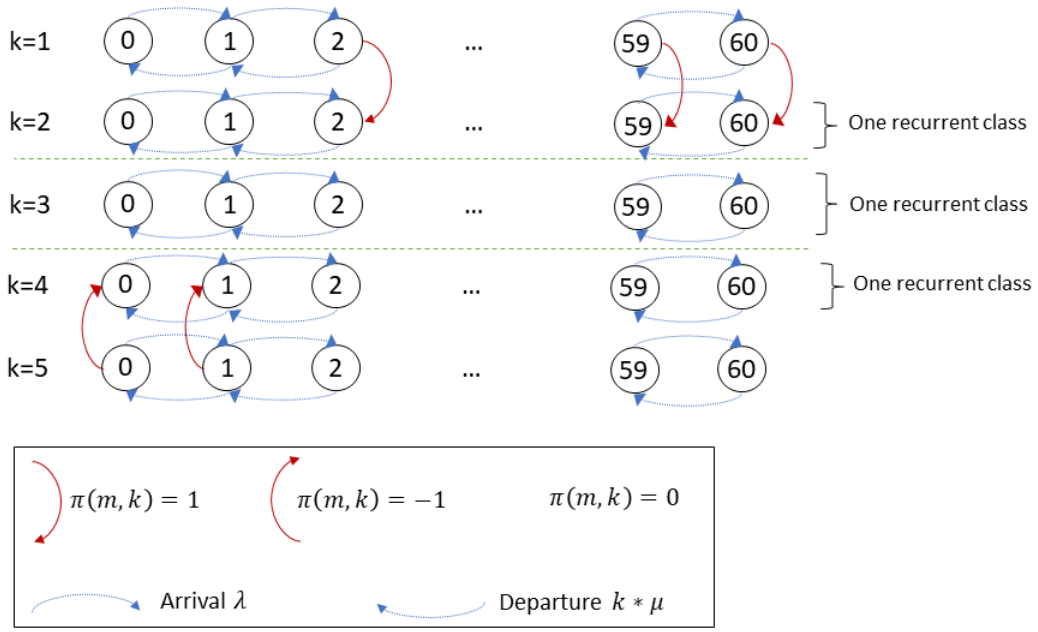

Figure 4: Example of an isotone hysteresis multichain MDP

B.2 Temporal behaviour of two approaches

We display two figures showing the different temporal behaviour between the two approaches: Markov chain and MDP. The first figure explains the temporal transition in the Markov chain approach while the second figure explains the temporal transition for MDP.

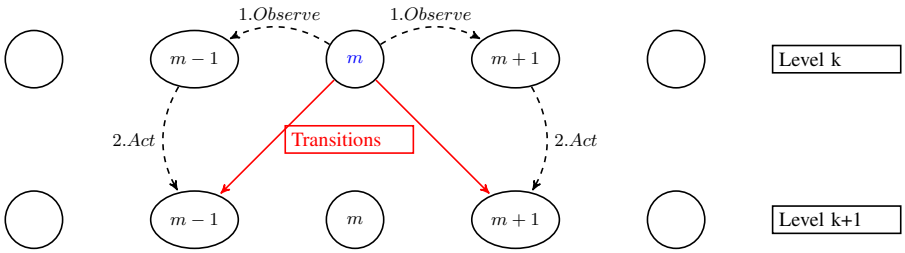

Figure 5: Behaviour of the agent in the Markov chain Model

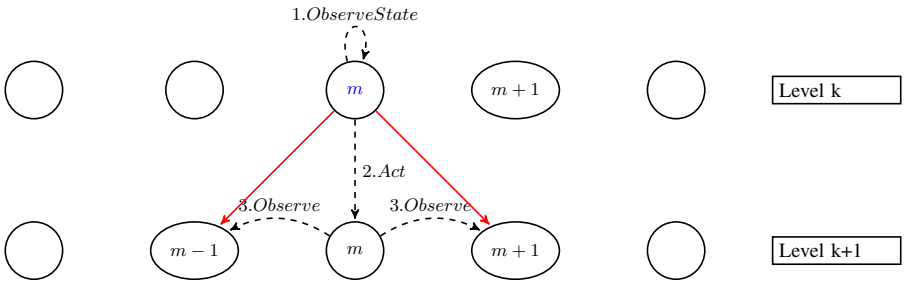

Figure 6: Behaviour of the agent in the MDP Model

B.3 Transformation of MDP policy into a threshold policy

We display an example of generated Markov chain by a policy q of the MDP agent. We show in this figure how we can transform the MDP policy into a threshold policy.

Example 3. *Example of the transformation of a MDP policy into a threshold policy with 3 virtual machines.*

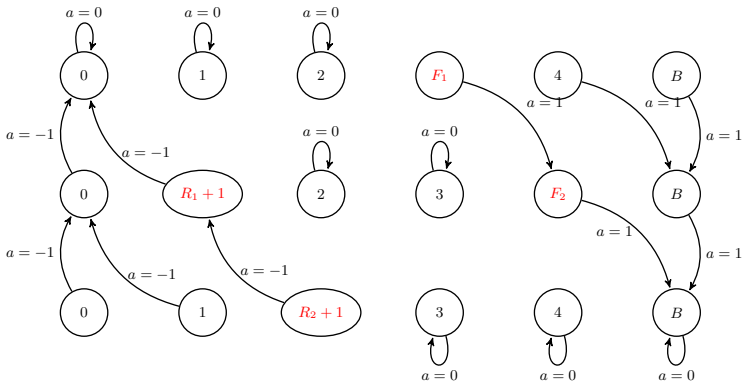

Figure 7: Example of the threshold policy transformation in a low scale scenario

The optimal threshold vector in this example would be: $L^* = [3, 4]$ *and* $l^* = [0, 1]$

B.4 Experimental results for concrete scenarios

We display several experimental (Section 7.3) results for concrete scenarios by comparing the behaviour of first activation thresholds and global costs when we take different parameterisation of the system (arrival rate λ, SLA metric N_{sla}).

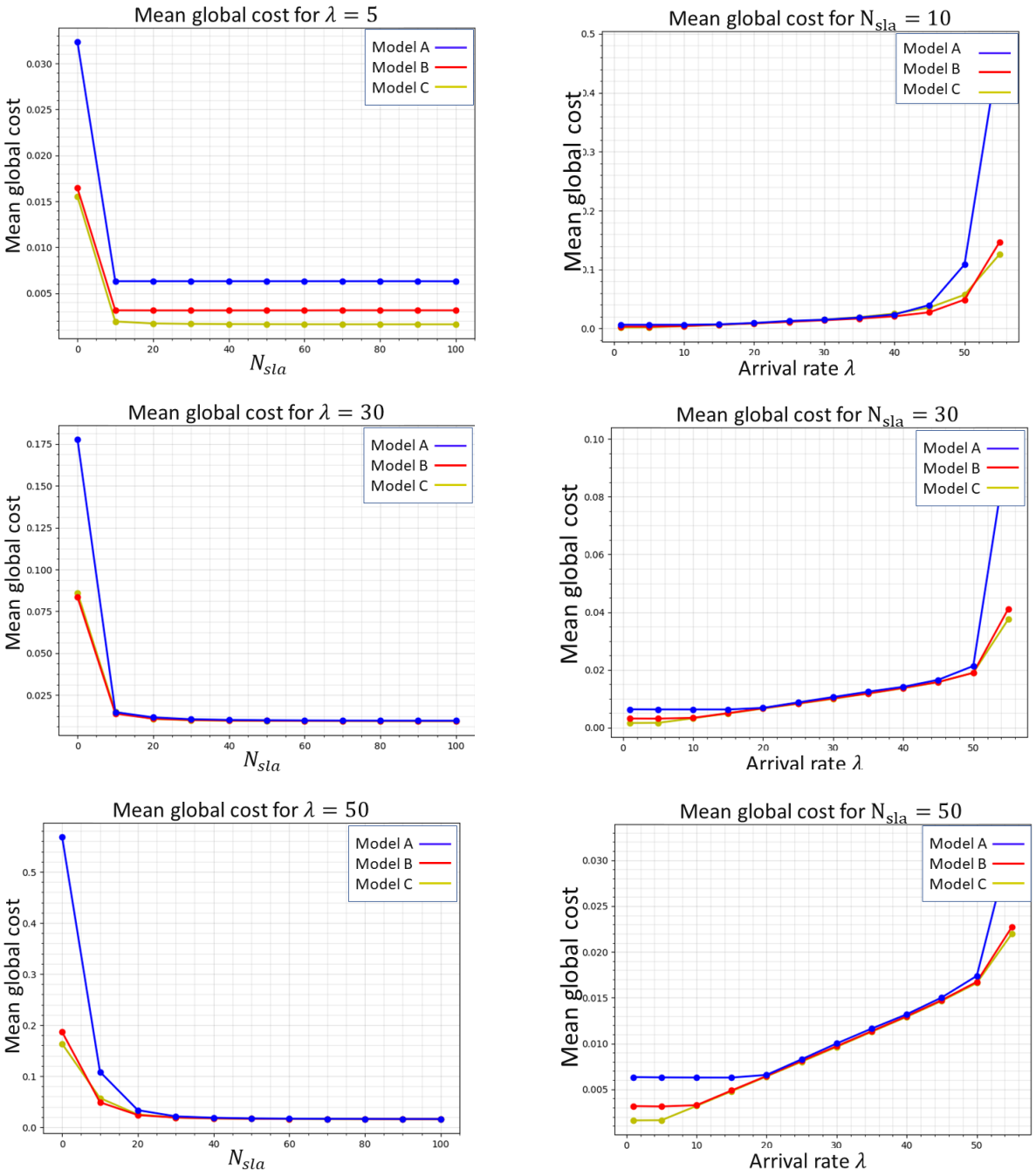

Figure 8: Mean global cost for the three models given a fixed Nsla or a fixed lambda

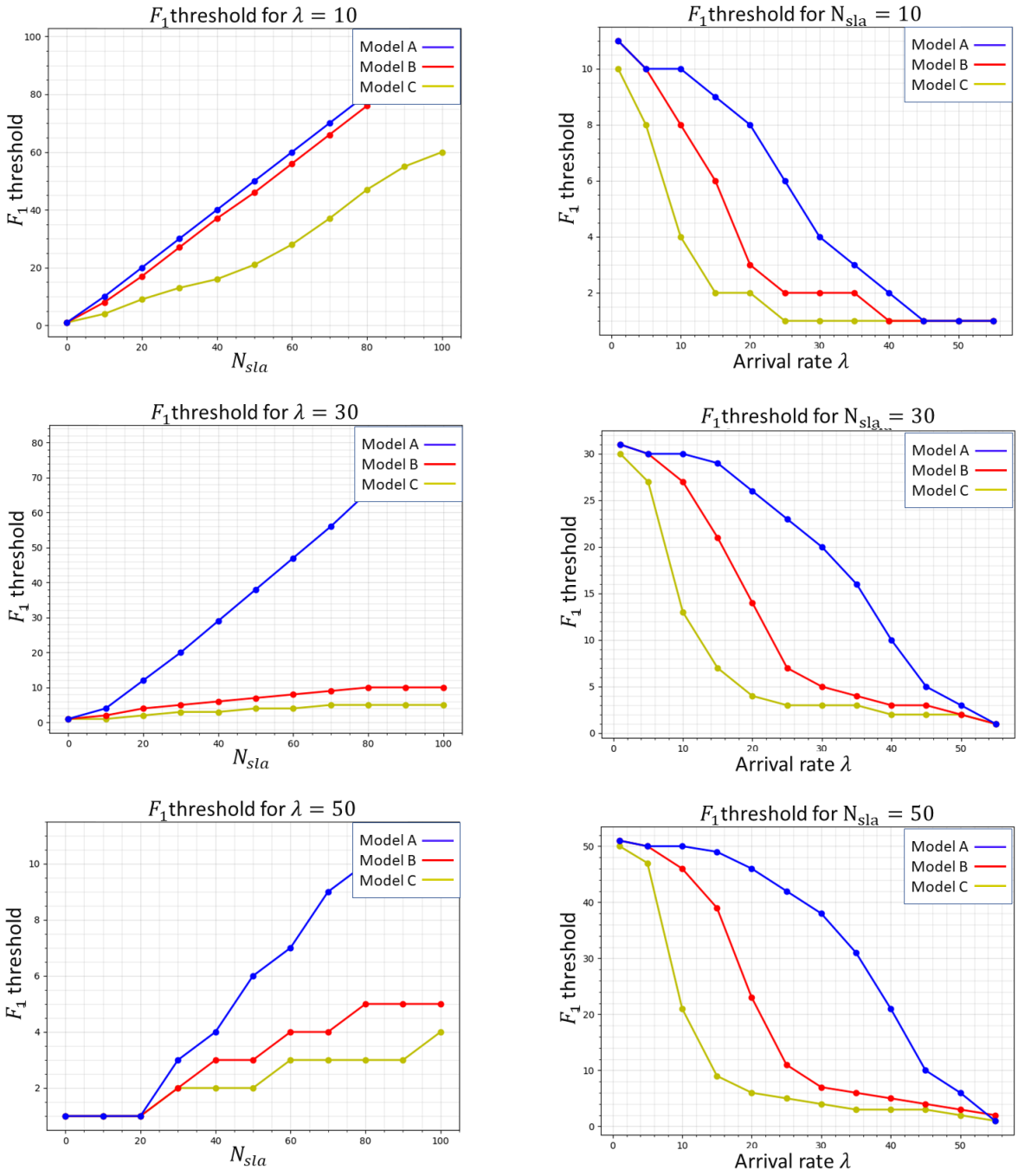

Figure 9: F1 thresholds for the three models given a fixed Nsla or a fix lambda

C Pseudo-Codes of algorithms

We present the pseudo-codes of heuristics described in section 4.3. To describe the different algorithms, we define the following notations:

- *solve*$[F, R]$: function that, for a given vector of thresholds, generates its associated Markov chain, computes its stationary distribution with the aggregation method then computes the mean cost $\overline{C}^{\pi}_{[F,R]}$.

- C^*: the solution of the optimisation problem and $[F, R]^*$ the optimal vector of thresholds.

Algorithm 1: Best per level

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$, the cloud system parameters
Output: $[F, R]^*, C^*$

Initialise a vector of thresholds $[F, R]_0$
Apply **solve**$[F, R]$ for the current vector
while *improvement=TRUE* **do**
    /* Activation Thresholds */
    for $k \in [1, K-1]$ **do**
        for F_k *respecting constraints* **do**
            solve$[F, R]$ and **store** the vector if improvement
    ---
    /* Deactivation Thresholds */
    for $k \in [1, K-1]$ **do**
        for R_k *respecting constraints* **do**
            solve$[F, R]$ and **store** the vector if improvement

Algorithm 2: Neighbourhood local search

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$, the cloud system parameters
Output: $[F, R]^*, C^*$

Initialise a vector of thresholds $[F, R]_0$
Apply **solve**$[F, R]$ for the current vector
while *improvement=TRUE* **do**
    /* We generate $\mathbf{V}$ by giving +1 or -1 to each elements of the vector, card$(\mathbf{V}) = 4(K-1)$ */
    Generate the neighbourhood $\mathbf{V}$ of $[F, R]$
    for $[F, R] \in \mathbf{V}$ **do**
        solve$[F, R]$ and **store** the vector if improvement

Algorithm 3: M/M/K/B approximation

Input: $\lambda, \mu, B, K, C_a, C_d, C_h, C_s, C_r$, the cloud system parameters
Output: $[F, R]^*, C^*$

Initialise a vector of thresholds $[F, R]_0$
Apply **solve**$[F, R]$ for the current vector
while *improvement=TRUE* **do**
    /* Activation Threshold 1st Level */
    First level k=1
    for $m \in [1, B]$ **do**
        Compute $\phi_1^A(m)$
        When $\phi_1^A(m) > 0$,BREAK, then F_1 = m if constraints respected
    /* Activation thresholds level k */
    for $k \in [2, K-1]$ **do**
        for $m \in [F_{k-1}+1, B]$ **do**
            Compute $\phi_k^A(m)$
            When $\phi_k^A(m) > 0$,BREAK, then F_k = m if constraints respected
    /* Deactivation Threshold 1st Level */
    First level k=1
    for $m \in [0, F_k - 1]$ **do**
        Compute $\phi_1^D(m)$
        When $\phi_1^D(m) \geq 0$,BREAK, then R_1 = m if constraints respected
    /* Deactivation thresholds level k */
    for $k \in [2, K]$ **do**
        for $m \in [R_{k-1}+1, F_k - 1]$ **do**
            Compute $\phi_k^D(m)$
            When $\phi_k^D(m) \geq 0$,BREAK, then R_k = m if constraints respected